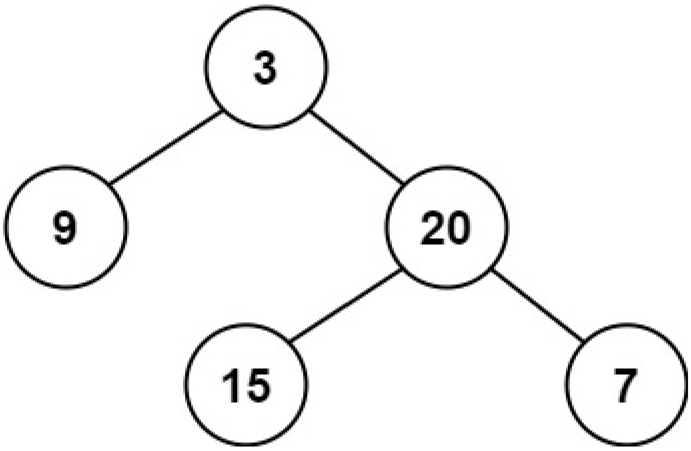
PRACTICE LINK::
<https://leetcode.com/problems/balanced-binary-tree/>

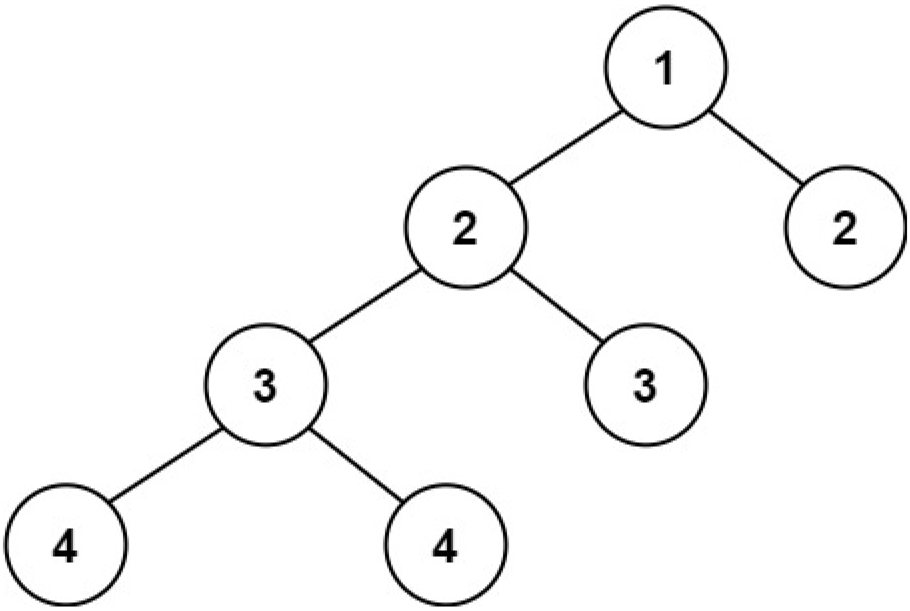
Given a binary tree, determine if it is height-balanced.

Example 1:



Input: root = [3,9,20,null,null,15,7]
Output: true

Example 2:



Input: root = [1,2,2,3,3,null,null,4,4]
Output: false
Example 3:

Input: root = []
Output: true

<<<<<<<-----CODE----->>>>>>>

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    bool isBalanced(TreeNode* root) {
        // If the tree is empty, we can say it’s balanced...
        if (root == NULL)  return true;
        // Height Function will return -1, when it’s an unbalanced tree...
                        if (Height(root) == -1)  return false;
                        return true;
            }
    // Create a function to return the “height” of a current subtree using recursion...
            int Height(TreeNode* root) {

        // Base case...

                        if (root == NULL)  return 0;

        // Height of left subtree...

                        int leftHeight = Height(root->left);

        // Height of height subtree...

                        int rightHight = Height(root->right);

        // In case of left subtree or right subtree unbalanced or their heights differ by more than
//‘1’, return -1...
                        if (leftHeight == -1 || rightHight == -1 || abs(leftHeight - rightHight) > 1)  return
-1;

        // Otherwise, return the height of this subtree as max(leftHeight, rightHight) + 1...
                        return max(leftHeight, rightHight) + 1;
    }
};
```